DATE: 4/01/97

CONTROLLING OFFICE FOR THIS DOCUMENT IS:
DIRECTOR, Army High Performance Computing
Research Center (AHPCRC)
Army Research Laboratory
Aberdeen, MD 21005

POC: Director (Tayn E. Tezduyar)

DISTRIBUTION STATEMENT A: Public release

Finite Element Flow Code Optimization on the Cray T3D

Sum 95

byStephon Saroff (AHPCRC-MSCI)

This is a second article on the porting and optimization of an AHPCRC finite element code on a Cray T3D. The first article appeared in the Winter/Spring 1995 issue of the AHPCRC Bulletin.

A Cray T3D system was installed at the Minnesota Supercomputer Center, Inc. (MSCI) in the Fall of 1993. At the present time, the system is configured with 512 processing elements and 32.8 Gigabytes of memory. Through a gift of time from MSCI and other arrangements, the AHPCRC has limited access to this system.

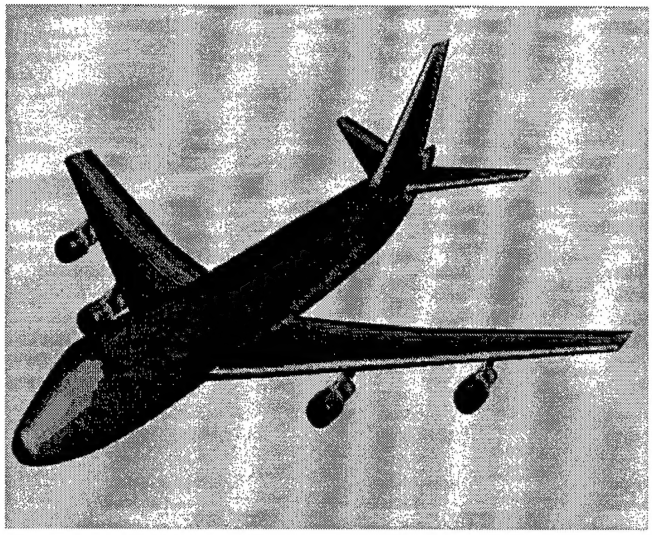

Figure 1. Flow past an aircraft modeled after 747. Pressure distribution on the aircraft surface at an inviscid flow condition of Mach 0.25.

The research group of Tayfun Tezduyar (Professor of Aerospace Engineering and Mechanics) ported a finite element flow solver, originally developed on and optimized for the Thinking Machines Corporation CM-5, to the Cray T3D. Since the Cray T3D provides more robust support for message passing models than for data parallel structures, the port required a translation of the code from data parallel CMF on the CM-5 to PVM message passing and nodal F77 on the T3D. The port involved manually decomposing the codes data structures and explicitly coding inter-processor communications. This was done by Marek Behr, an Assistant Professor at the AHPCRC, and Shahrouz Aliabadi, a Research Associate at the AHPCRC, and was described in the Winter/Spring 1995 Bulletin. Examples of compressible flow simulations, carried

out by the 1995 Summer Institute students, are shown in Figures 1 and 2.

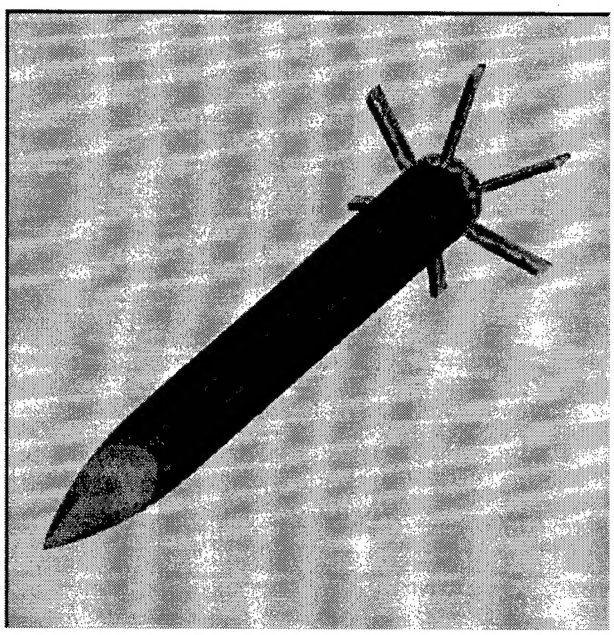

Figure 2. Flow past a missile geometry provided by ARL. Pressure distribution on the missile surface at an inviscid flow condition of Mach 2.5

The node level computations of the initial port of the code performed relatively poorly. This was not surprising. Although CM-5 and T3D processing nodes have similar nominal performance specifications (128 Mflops/node on the CM-5 compared to 150 Mflops/node on the T3D), their architectures are very different.

Table 1 provides the timing results following the initial port. Time to completion for the entire code is shown along with execution times of each of the major routines within that code. These major routines collectively account for over 95% of the total execution time.

	T3D	**CM-5**
time to completion	3538.52	2097.81
blkuvp	2052.24	991.44
get_jac_turb	315.56	169.68
gen_matrix_vector_mult_noadd	455.89	203.03
scatter	358.16	259.69
gather	207.48	200.35

Table 1. Initial comparison of CM-5 and T3D timings (execution time in seconds on 32 nodes).

This author, with the participation of analysts from Cray Research and MSCI, undertook an effort to improve the nodal

performance of the code (i. e. Reduce the computation time required per processing element).

An examination of the incompressible flow code with non-matrix-free solver disclosed that the routines blkuvp, get_jac_turb and gen_matrix_vector_mult_noadd are compute intensive routines. The routines scatter and gather are inter-processor communications routines. It was decided to concentrate optimization efforts on blkuvp and get_jac_turb. The gen_matrix_vector_mult_noadd is a matrix algebra routine and should be optimized through calls to math library functions.

The optimization effort focused on issues of efficient register and cache usage, primarily through index reordering and use of temporaries. This is required because the T3D processing element uses an EV-4 Alpha microprocessor which has only one port to memory, a limited number of floating point registers (32), and only one cache (1K 64-bit words).

In the original CM-5 version of blkuvp, the bulk of the execution time was spent in a series of loops nested three deep, which (translated from CMF to F77) were of the form:

```
do na = 1,nen
   na0 = (na-1) * ndf
   nau = na0 + udf
   nav = na0 + vdf
   . . .
   do nb = 1,nen
      nbp = (nb-1) * ndf + pdf
      sh0sh0(ie) = sq(0,na,iq) * sq(0,nb,iq)
      sh1sh0(ie) = sh(1,na,ie) * sq(0,nb,iq)
      sh2sh0(ie) = sh(2,na,ie) * sq(0,nb,iq)
      . . .
      tmp1(ie) = - sh1sh0(ie) * eff0(ie) + gua(ie) * sh(1,nb,ie)
* effs(ie)
      tmp2(ie) = - sh2sh0(ie) * eff0(ie) + gua(ie) * sh(2,nb,ie)
* effs(ie)
      s(nau,nbp,ie) = s(nau,nbp,ie) + tmp1(ie)
      s(nav,nbp,ie) = s(nav,nbp,ie) + tmp2(ie)
      . . .
```

It was observed that there were a significant number of arrays of temporaries, for example:

```
do ie = 1,nec
   sh1sh0(ie) = sh(1,na,iq) * sq(0,nb,iq)
   sh2sh0(ie) = sh(2,na,ie) * sq(0,nb,iq)
```

While this is a necessary technique in CMF data parallel programming, it is not optimal on scalar or vector processors. Instead, the temporary arrays were replaced with scalar temporaries. This allows the compiler to keep values in registers and avoid cache misses.

```
do ie = 1,nec
   mysh1na = sh(1,na,ie)
   mysh2na = sh(2,na,ie)
   mysq0nb = sq(0,nb,iq)
   mysh1sh0 = mysh1na * mysq0nb
   mysh2sh0 = mysh2na * mysq0nb
```

This change reduced the time to completion of the main loop of "blkuvp" by more than 50%. To ensure more efficient memory access, and to avoid cache misses, the loop order was then changed to allow the right most array index to be associated with the outermost do loop (the index ie was generally the right most index).

```
do ie = 1,nec
   do nb = 1,nen
      nbp =(nb - 1) * ndf + pdf
      do na = 1,nen
         na0 = (na-1) * ndf
         nau = na0 + udf
         nav = na0 + vdf
         mysh1na = sh(1,na,ie)
         mysh2na = sh(2,na,ie)
         mysq0nb = sq(0,nb,iq)
         mysh1nb = sh(1,nb,ie)
         mysh2nb = sh(2,nb,ie)
         mysh1sh0 = mysh1na * mysq0nb
         mysh2sh0 = mysh2na * mysq0nb
         . . .
         tmp1 = - mysh1sh0 * eff0 + gua * mysh1nb * effs
         tmp2 = - mysh2sh0 * eff0 + gua * mysh2nb * effs
         s(nau,nbp,ie) = s(nau,nbp,ie) + tmp1
         s(nav,nbp,ie) = s(nav,nbp,ie) + tmp2
         . . .
```

	Orignal Code	**Using Temporaries**	**Both Tempories and Reordering**
blkuvp (all)	2052.24	1275.32	1094.70
lkuvp (main loop)	1859.97	744.00	616.22

Table 2. Improvements in "blkuvp" performance (execution time in seconds on 32 nodes).

This restructuring resulted in an additional performance improvement in the main loop of blkuvp, whereby the execution time was reduced to 33% of its pre-optimization time. The results are shown in Table 2.

Similar steps were undertaken for "get_jac_turb". That is, loop order was reversed and temporaries replaced.

time to completion	2164.81	2097.81
blkuvp	880.96	991.44
get_jac_turb	151.45	169.68
gen_matrix_vector_mult_noadd	450.72	203.03
scatter	326.03	259.69
gather	206.69	200.35

Table 3. Final comparison of CM-5 and T3D timings (execution time in seconds on 32 nodes).

The purpose of using scalar temporaries was to improve cache and register usage, thus reducing the amount of costly memory access. Each scalar value retained by the compiler in a register eliminates the need to access either cache (at 3 cycles per access) or main memory (at 22-39 cycles per access) for that value. However, since there are far more temporaries than registers (resulting in the spilling of some temporaries to cache), the improvement of the codes performance cannot be entirely attributed to register usage. This can be seen from the fact that significant gains in performance could be produced without a significant reduction in the number of private loads. With the substantial time saved by loading values from cache rather than from memory, a significant portion of the performance increase resulted from enabling the compiler to maintain cache integrity (i.e. keeping values in the cache, rather than having to read them from main memory). The final comparison is shown in Table 3.

Larger Scale Simulation Capability for Ram Air Parafoils is Achieved on the Cray T3D

Sum 95

by Shahrouz Aliabadi (AHPCRC-UMN)

Realistic numerical simulations of complex flows require very high grid resolutions. Using highly refined meshes in numerical simulations not only improves the accuracy of the solution but also may predict some small-scale physical characteristics such as high frequency modes in turbulence. These physical characteristics might be absent in a solution obtained from a coarser mesh. The major barriers involved in these large-scale computations are memory and CPU time. Efficient algorithms and advanced hardware with sufficient computational power are the keys to removing the barriers involved in large scale simulations.

The author of this article, a member of the research team of Tayfun Tezduyar, Professor of Aerospace Engineering and Mechanics, is carrying out very large scale computations to predict the flow field and the dynamics of ram air parafoils. The steady-state performance of these parafoils is simulated using a finite element mesh consisting of 9,741,930 nodes and 9,595,520 hexahedral elements. A coupled nonlinear system of equations with 38,391,715 unknowns is solved at every time step. The Cray T3D with 512 processors and highly optimized finite element flow solvers were utilized to solve this problem. The AHPCRC has, under special arrangements, limited access to this Cray T3D owned and operated by the Minnesota Supercomputer Center, Inc.

The finite element flow solvers used are based on state-of-the-art stabilized formulations which maintain their numerical stability and accuracy even at high Mach numbers and high Reynolds numbers. The formulations are also capable of handling problems with moving boundaries and interfaces. The coupled equation systems arising from the finite element discretization are solved using matrix-free iteration techniques. The matrix-free iterations totally eliminate the need to store the element-level matrices corresponding to the left-hand-side matrix. This significantly reduces the memory requirements in finite element computations.

Figure 1. Air flow past a large ram air parafoil is simulated on the Cray T3D with 512 processors. The finite element mesh used in this computation consists of 9,741,930 nodes and 9,595,520 elements which results in 38,391,715 coupled equations. The picture shows the computed flow past a large ram air parafoil during a steady-state gliding descent at two degrees angle of attack and a Reynolds number of 10 million. Algebraic turbulence models are utilized for this high Reynolds number flow. The colors depict the pressure distibution on the parafoil surface.

Dealing with the large data sets involved in these computations at this scale was a major undertaking. Special techniques were used to display the pressure distribution on the parafoil surface. The visualization of such large data sets is still an open issue in flow simulations involving complex, real-world problems, and needs further research.